PID Playback Lib

A library for playing a trajectory on a force feedback device

USER MANUAL

V1.1

Table of Contents

1. Release Info 3
   1.1. Intro 3
   1.2. New features for v1.1 3
   1.3. Units 3
2. Safety Information 3
3. How to use the library 4
4. Useful Functions and Variables 4
   4.1. Playback Constructor 4
      4.1.1. Using the Controllers 4
         Creating the Controllers 4
         Reset the Controller State 5
      4.1.2. Playback Modes 5
         Playing the Gesture 6
         Playback Options 7
         Turning off the playback 8
         Sticking to a Point 8
         Creating, Saving and Manipulating the Path 9
5. Sample MFC Record and Playback 10

1. Release Info

1.1. Intro

This documentation describes the use of PID Playback Lib version 1.1 - A library for playing a trajectory on an active force feedback device. This code has been developed by Andrew Crossan (ac@dcs.gla.ac.uk) and John Williamson (jhw@dcs.gla.ac.uk) at the University of Glasgow.

The path following system was developed by Andrew Crossan and John Williamson. The PID controller was developed by John Williamson. Feel free to use and distribute, but please do not change the header information in Playback.h and PID.h. All comments or suggested additions welcome.

1.2. New features for v1.1

- The major update has been the inclusion of a constrain mode where the using is constrained to a trajectory. **StartConstrain** can be called (where StartPlayback was previously called) to constrain the user to moving forwards along a predetermined path. While controller forces will hold the device to the path, there are no forces to drive the user along the path.
- Create a PD controller with preset values with **CreatePDControllersForOmni**
- An extra function added to stick the phantom to the current position using **StickToThisPoint**
- To get the current position of the playback, you can now use the function **GetPlaybackPos**
- **SetIntegratorForceCap** is added to set the maximum force value that the integrator can reach in x, y and z.
- **SaveGestureFile** has been added to allow saving the currently stored trajectory

1.3. Units

All default controller and playback settings are using the units **mm** for distance and **seconds** for time. The easiest way to get around this is to pre-scale your trajectory positions as you are loading them into the playback controller. If this isn't possible, the controller values should be re-tuned to the units that the program requires.

2. Safety Information

This is an early version of the library so there **WILL BE BUGS**. This library is no way intended for any safety critical work. A force saturation level can be specified for the controller during creation. This value should **ALWAYS** be set at a safe value where the user can overpower the controller.

This library is to be used at your own risk. We take no responsibility for any injuries caused during playback.

3. How to use the library

Include the playback.lib file in your C++ project. Make sure that the .h files are accessible to the project. Note

To playback a trajectory on the haptic device using the controller:

1. Include the file playback.h in the file where the controller is to be used.
2. Add a variable of type Playback. Ie Playback pb;
3. Initialise the controller using one of the CreateController functions. Ie. pb.CreateController(…) (For Sensable OMNI or Premium 1.5 Phantom devices, pre-tuned controllers can be created by using CreateControllersForOmni() or CreateControllersForPremium15() instead).
4. Add a series of points for the trajectory using AddWaypoint. Ie. pb.AddWaypoint(pt).
5. Set the controller playing with StartPlayback.
6. At every servo loop update, call GetForce to get the controller force. GetForce will return false when the playback is finished

4. Useful Functions and Variables

4.1. Playback Constructor

The Playback object constructor takes no parameters

```
Playback();
```

4.1.1. Using the Controllers

Creating the Controllers

To Initialize a new PID controller with specific values. The controllers must be created once only, but before the playback starts

```
void CreateControllers(double p, double i, double d, double f,
      double out_filter, double gain, double sat_low, double sat_high,
      double dead_zone);
```

Parameters:
p: proportional control gain
i: integral control gain
d: derivative control gain
f: input filter (for deriv. estimation)
out_filter: output filtering
gain: overall gain
sat_low: minimum force saturation point
sat_high: maximum force saturation point
dead_zone: A control dead zone. The error distance below which no controller forces are applied.

PID Preset controllers

The following functions create PID controllers for standard devices. Currently parameters have been preset for Sensable's Phantom OMNI device and Phantom 1.5 device.

To initialise a PID controller with preset Phantom omni parameters

```
void CreateControllersForOmni();
```

(p = 0.3, i = 0.0005, d = 20.0, f = 0.09, out_filter = 0.09, gain = 0.2, sat_low = -3.0, sat_high = 3.0, dead_zone = 0.0)

To initialise with preset Phantom Premium 1.5parameters

```
void CreateControllersForPremium15();
```

(p = 0.3, i = 0.00005, d = 30.0, f = 0.09, out_filter = 0.09, gain = 0.2, sat_low = -1.5, sat_high = 1.5, dead_zone = 0.0)

PD Preset Controllers

The following functions create PD controllers for standard devices. Use PD instead of PID when force wind up might be a problem. Ie. if the user resists the motion the I component of the PID controller will increase the output force to overcome the resistance and thus build up the force continually until over coming the resistance. By setting the I component of the controller to zero we avoid this wind up at the cost of less ability to overcome obstacles.

Currently parameters have been preset for Sensable's Phantom OMNI device.

```
void CreatePDControllersForOmni();
```

Initialise PD controller with preset Phantom omni parameters
p = 0.3, i = 0.0, d = 20.0, f = 0.09, out_filter = 0.09, gain = 0.2, sat_low = -3.0, sat_high = 3.0, dead_zone = 0.0

Reset the Controller State

To reset the state of the controller

```
void ResetControllerState();
```

4.1.2. Playback Modes

The header file contains the definition

```
enum Mode
{
	PID_IDLE                        = 0,
	PID_STICK_TO_POINT              = 1,
	PID_MOVE_TO_STICK_TO_POINT      = 2,
	PID_PLAY                        = 3,
	PID_MOVE_TO_PLAY_POINT          = 4,
	PID_PAUSE                       = 5,
	PID_CONSTRAIN                   = 6,
	PID_MOVE_TO_CONSTRAIN_POINT     = 7,
};
```

The MOVE_TO_* modes (PID_MOVE_TO_STICK_TO_POINT, PID_MOVE_TO_PLAY_POINT and PID_MOVE_TO_CONSTRAIN_POINT) are used initially when StickToPoint is called or, if using absolute positioning, if StartPlayback or StartConstrain is called. The MOVE_TO_* modes will move the user's cursor smoothly to the start position for the trajectory before automatically switching the appropriate mode once within the error threshold of the playback. The one exception is when StartPlayback is called when playing back at sample rate where the mode is switched straight to PID_PLAY.

```
Mode GetMode();
To get the current mode of playback
```

Playing the Gesture

Start playback is called to set the trajectory to play. You must still call GetForce every servo loop update to get the force of the playback either specify a trajectory explicitly by passing in a list of points or if no list is provided, the pre-built (with AddWaypoint) trajectory is used

Dragging the User Around a Path

To playback currently stored waypoints, call StartPlayback. If playing in absolute positioning mode, the controller is initially set to PID_MOVE_TO_PLAY_POINT until the cursor is within the error threshold of the controller. The mode will then automatically switch to PID_PLAY and the gesture will start playing. GetForce can then be used at every servo loop update to get the force required to control the device.

```
void StartPlayback(double curPos[3]);
Where curPos is the current x, y, z position of the device. Te
waypoints are added to the path by using the AddWaypoint function
described below
```

To playback specify the waypoints at the time of the function call, use

```
void StartPlayback(PtsList *pt, double curPos[3]);
Where curPos is the current x, y, z position of the device. And pt is a
list of x, y, z waypoint positions. The type PtsList is defined in
playback.h as

NOTE: using StartPlayback(PtsList *pt, double curPos[3]) will overwrite
any trajectory specified with AddWapoint

struct PtsList {
   double xPos, yPos, zPos;
   PtsList *prev, *next;
};
```

Constraining the User to a Path

To constrain the user to the currently stored waypoints, call StartConstrain. If playing in absolute positioning mode, the controller is initially set to PID_MOVE_TO_CONSTRAIN_POINT until the cursor is within the error threshold of the controller. The mode will then automatically switch to PID_CONSTRAIN and the gesture will start playing.

```
void StartConstrain(double curPos[3]);
During playback, at every servo loop update call
```

Updating the Device Forces

GetForce is the key to controlling the device. This function is to be called at every servo loop update. It will check the mode the controller is in and work out current forces if required to control the device.

```
bool GetForce(double pt[3], double force[3]);
Where pt[3] is the current device x, y, z position. After the call,
force will contain the force calculated by the controller. Returns
false only if the playback trajectory is finished.
```

The Current Playback Position

At any time, the current position that the controller is aiming to reach can be determined by GetPlaybackPos. This function returns true unless the Controller is in PID_IDLE mode. If not idle, the function returns in the parameter pPos, the current position that the controller is aiming for

```
bool GetPlaybackPos(double pPos[3]);
Returns true only if not IDLE. In pPos, the current x, y, z playback
position or stick point position will be returned
```

Playback Options

The following playback options are available.

Play Speed Mode – The controller can be set to either playback at the recorded speed or at a constant speed specified by **BEAD_SPEED.** At the recorded speed, the trajectory will move on by one sample point every update. Note that if the user resists the movement, the shape of the trajectory will degrade as the playback position will move on even if the user holds the device still. If playing at a constant speed, the shape of the trajectory will be maintained by only allowing the playback position to move if the user is within a certain distance. The deaulf mode is specified in Playback.h through **PLAY_AT_SAMPLE_RATE.** If true, by default, the trajectory will be played back at the recorded rate. If false, the trajectory will be played back at a constant rate specified by **BEAD_SPEED.**

```
void SetPlayAtSampleRate(bool pasr);
sets the controller to playback one sample per update. If false,
playback rate is controller through SetPlaybackSpeed. NOTE... this
means that if the user resists moving, the gesture will still continue
```

The Play speed mode can be determined by calling

```
bool GetPlayAtSampleRate();
Returns true if the controller is set to playback at the sample rate
```

Set the Playback Speed – This sets the speed that the trajectory plays back if not playing at the sample rate. BEAD_SPEED in Playback.h gives

the default speed of playback in mm/update. The playback speed can be changed by calling.

```
void SetPlaybackSpeed(double v);
```

Positioning mode – specify the positioning mode (relative or absolute) using UseAbsolutePositioning. If false, the playback trajectory will be played back relative to the current deice position. If true, the trajectory co-ordinates will be played back as absolute position within the environment. Default positioning mode is specified by the constant **ABSOLUTE_POSITIONING** defined in playback.h

```
void UseAbsolutePositioning(bool absPos);
absPos == True to use absolute Positioning for gesture
absPos == False to start gesture at the current position
```

Error Threshold – The error threshold is the distance that the device has to get to the current position in the trajectory before the trajectory is moved to the next sample point (when not playing back at sample rate). SetErrorThreshold and GetErrorThreshold set and get this value respectively in mm. Default is 10mm.

```
void SetErrorThreshold(double e);
double GetErrorThreshold();
```

Integrator Cap – The integrator cap is the maximum for that the 'i' component of the controller can build up to in x, y, and z. Without setting this value the total force will still saturate at the maximum force level set when creating the controllers, but the 'i' component will continue to build causing a force wind up effect. You can set the integrator cap for the x, y and z axes using SetIntegratorForceCap.

```
void SetIntegratorForceCap(double maxforce);
```

Turning off the playback

To set the playback state to idle (ie. stop all forces) call

```
void SetIdle();
```

You can also toggle between pause and play. Pause will stop the gesture (if playing) but will not return it to the start of the playback. Calling TogglePauseGesture while paused will resume the playback from the current waypoint.

```
void TogglePauseGesture();
```

Sticking to a Point

This mode will attempt to hold the device at a point specified by the programmer

```
void StickToPoint(double curPos[3], double pt[3]);
```

curpos is the current x, y, z position of the device. pt is the point to hold the device at. This is called ONCE to set the mode with

GetForce (previously specified in the 'Playing the Gesture' section) called at every update to return the device force required to stick to the point as when playing the gesture. Note that GetForce will return true in this mode until the mode is changed.

Alternative, the device can be stuck to the current position

```
void StickToThisPoint(double curPos[3]);
Sticks the cursor to the currentPosition specified by curPos
```

Creating, Saving and Manipulating the Path

The path is stored in a PtsList as x, y, z co-ordinates. To create an empty list or clear a list at any point, call

```
void ClearPath();
To add a waypoint to the end of the current trajectory, call

void AddWaypoint(double pt[3]);
where pt specifies the x, y, z position
When calling StartPlayback(double curPos[3]), addWaypoint is used to
pre-specify the path.
```

To move to the start of the trajectory, call

```
void MoveToStartOfGesture();
```

To load a trajectory from a file call

```
bool LoadGestureFile(char *fName);
where fName is the path and name of the file
returns true if the file is found and false otherwise
```

File Format

```
time1, pt1_x, pt1_y, pt1_z
time2, pt2_x, pt2_y, pt2_z
time3, pt2_x, pt2_y, pt2_z
.......
```

After a call to LoadGestureFile, StartPlayback(double curPos[3]) will play the trajectory that has been loaded from the file.

NOTE: The timing data is currently ignored. This will be sorted in a future update

To save the current gesture to a file use SaveGestureFile. Returns true if the operation succeeds and false otherwise.

File Format

```
time1, pt1_x, pt1_y, pt1_z
time2, pt2_x, pt2_y, pt2_z
time3, pt2_x, pt2_y, pt2_z
.......
```

```
bool SaveGestureFile(char *fName)
NOTE: The timing data is currently NOT implemented. Times saved will all
be 0.0. This will be sorted in a future update
```

5. Sample MFC Record and Playback

```
// RecordAndPlaybackDlg.cpp : implementation file
//

#include "stdafx.h"
#include "RecordAndPlayback.h"
#include "RecordAndPlaybackDlg.h"
#include ".\recordandplaybackdlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


/* An example program usign the playback library with the
 * Phantom OMNI and OpenHaptics */

// Phantom stuff
HDCallbackCode HDCALLBACK PhantomCallback(void *data);
HHD hHD;

// to contain the state of the phantom buttons
// true for button down
bool pB1Down = false, pB2Down = false;


CRecordAndPlaybackDlg *dlg;

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	enum { IDD = IDD_ABOUTBOX };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()


// CRecordAndPlaybackDlg dialog



CRecordAndPlaybackDlg::CRecordAndPlaybackDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CRecordAndPlaybackDlg::IDD, pParent)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

```
void CRecordAndPlaybackDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CRecordAndPlaybackDlg, CDialog)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	//}}AFX_MSG_MAP
	ON_WM_CLOSE()
END_MESSAGE_MAP()


// CRecordAndPlaybackDlg message handlers

BOOL CRecordAndPlaybackDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	dlg = this;

	// Omni Initialisation
	hHD = hdInitDevice(HD_DEFAULT_DEVICE);
	hdEnable(HD_FORCE_OUTPUT);

	hdStartScheduler();
	hdScheduleAsynchronous(PhantomCallback,(void*)0,HD_DEFAULT_SCHEDULER_PRIORITY);
	hdGetDoublev(HD_CURRENT_POSITION, pPos);

	// playback initialisation
	pb = new Playback();
	pb->SetPlaybackSpeed(100);
	pb->CreateControllersForOmni();

	// set the error threshold in mm
	pb->SetErrorThreshold(8);

	// avoid force wind up above 2N
	pb->SetIntegratorForceCap(2);

	// used in the Omni callback to set the initial state of the playback
	firstTime = true;

	// this line will ensure playback of one sample point per update
	//pb->SetPlayAtSampleRate(true);
```

```
	// this will set the gesture to start playing from the position it
	// was recorded from
	//pb->UseAbsolutePositioning(true);

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CRecordAndPlaybackDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CRecordAndPlaybackDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()),
0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this function to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CRecordAndPlaybackDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}

// PHANToM Stuff
HDCallbackCode HDCALLBACK PhantomCallback(void *data)
{
	HHD hHDlocal = hdGetCurrentDevice();
	hdBeginFrame(hHDlocal);

	//retrieve the position of the end-effector.
	hdGetDoublev(HD_CURRENT_POSITION, dlg->pPos);
	HDdouble force[3];

	HDdouble vel[3];
	hdGetDoublev(HD_CURRENT_VELOCITY, vel);

	// test button input
```

```
        HDint buttons;
        hdGetIntegerv(HD_CURRENT_BUTTONS,&buttons);

        if(dlg->firstTime){
                dlg->firstTime = false;

                double pt[3] = {0.0, 20.0, 0.0};
                dlg->pb->StickToPoint(dlg->pPos, pt);
        }

        if(buttons == 1){ // then button 1 down
                if(pB1Down == false){ // then button down event
                        // clear the current trajectory and set the state to idle
                        // so no forces from the controller to disuprt the recording
                        dlg->pb->ClearPath();
                        dlg->pb->SetIdle();

                }

                pB1Down = true;

                // add to the current trajectory
                dlg->pb->AddWaypoint(dlg->pPos);

        }
        else if(buttons == 2){ // then button 2 down
                if(pB2Down == false) // then button down event
                        dlg->pb->StartPlayback(dlg->pPos);
                        // to enter constrain mode, switch the above line to
                        //dlg->pb->StartConstrain(dlg->pPos);

                pB2Down = true;
        }
        else if(buttons == 0){

                if(pB1Down == true || pB2Down == true){ // then button up event

                        pB1Down = false;
                        pB2Down = false;

                        // hold the phantom a central position
                        double pt[3]; pt[0] = 0.0; pt[1] = 20.0; pt[2] = 0.0;

                        dlg->pb->StickToPoint(dlg->pPos, pt);
                }

        }


        // given the position, calculate a force
        // controller force is returned through the parameter force
        dlg->pb->GetForce(dlg->pPos, force);

        // set the force to the device
        hdSetDoublev(HD_CURRENT_FORCE, force);
        // flush the force
        hdEndFrame(hHDlocal);
        // run at every servo loop tick.

        return HD_CALLBACK_CONTINUE;
}

void CRecordAndPlaybackDlg::OnClose()
{
        // stop the playback
        pb->SetIdle();
        pb->ClearPath();

        // stop the phantom scheduler
        hdStopScheduler();
```

```
	CDialog::OnClose();
}



// RecordAndPlaybackDlg.h : header file
//

#pragma once

#include "Playback.h"
// Omni Stuff
#include <HD/hd.h>
#include <HDU/hduVector.h>

// CRecordAndPlaybackDlg dialog
class CRecordAndPlaybackDlg : public CDialog
{
// Construction
public:
	CRecordAndPlaybackDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
	enum { IDD = IDD_RECORDANDPLAYBACK_DIALOG };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support



// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()

public:
	bool firstTime;

	Playback *pb;
	HDdouble pPos[3];

	afx_msg void OnClose();
};
```